GUMP: Adapting Client/Server Messaging Protocols into Peer-to-Peer Serverless Environments

Robert N. Lass
Department of Computer Science
Drexel University
urlass@cs.drexel.edu

Joe Macker
Naval Research Lab
Washington DC, USA
joseph.macker@nrl.navy.mil

David Millar
Department of Computer Science
Drexel University
david.w.millar@gmail.com

Ian Taylor*
School of Computer Science
Cardiff University, Cardiff, UK
Ian.J.Taylor@cs.cardiff.ac.uk

ABSTRACT

In this paper we present a generic environment for creating message-oriented server-side proxies to support adaptation from TCP transport-oriented client-server sessions to many-to-many peer-to-peer networking environments more suitable for deployment in dynamic wireless networks, capable of multicast forwarding. At its input, GUMP provides an interface for exposing network server implementations in order to allow existing GUI applications to connect to GUMP. At the back-end, GUMP's generic service discovery and multicast interfaces allow access to multiple implementations, enabling the discovery of necessary services on the network, maintenance of the network state, and transport of messages amongst peers, for tuning to a specific network environment. At the heart of GUMP, there is a mechanism for selecting a server-side proxy implementation for a given messaging protocol, allowing multiple proxies to co-exist and run time adaption of the system. As a primary example and use case, we show how GUMP has been used to implement an XMPP proxy allowing existing off-the-shelf XMPP client software to dynamically create and operate multi-user chat sessions in a serverless network environment. This resulting proxy integration demonstrates the power of GUMP in its ability to adapt between different methods of input using either HTTP or TCP oriented server systems, the use of its different discovery subsystem bindings (SLPv2 and JmDNS), and its support for multicast architectures. GUMP therefore allows a single messaging protocol server-side implementation to be dynamically adapted to suit a particular distributed wireless deployment environment at run time.

*Corresponding Author

BADS'10, June 11, 2010, Washington, DC, USA.

Categories and Subject Descriptors

ERCIM Session [**Adaptive service discovery and composition**]

General Terms

Management, Reliability, Design

Keywords

XMPP, Serverless chat, WS-Notification, SLP, JmDNS, Peer-to-Peer, Multicast, NORM

1. INTRODUCTION

A number of applications and XML-based standardized messaging protocols typically assume dependence on TCP transport for achieving reasonable reliability across wide area networks. Historically, UDP and transport enhancements built above UDP have a more restricted portfolio of Internet usages, such as real-time applications requiring low latency, real-time delivery e.g. video or audio streaming. Whereas strict server centralization and TCP-oriented transport is often suitable in more fixed network environments, the design issues can be quite different in highly dynamic and disruptive network environments (e.g., localized mobile wireless domains). In more dynamic wireless networking and mobile ad hoc networks (MANETs) [1], this is even more apparent, as congestion control can often overreact to temporal disruptions (e.g., wireless link errors, collisions, routing dynamics) that are not indicative of congestion. Dynamic, wireless networking environments are also often less suitable for centralized server deployment due to potential server disruptions and mobility. Another point is that multicast or group-oriented network transport and delivery is often more suitable in wireless environments and therefore collaborative applications may often employ forms of UDP-based transport protocols or transport Performance Enhancing Proxies (PEPs), rather than pure end-to-end TCP.

XMPP [2, 3], a set of open XML technologies for presence and real-time communication developed by the Jabber open-source community, is an example of a standardized technology that has mostly been developed for TCP and client/server operation scenarios. XMPP is a good exam-

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUN 2010		**00-00-2010 to 00-00-2010**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
GUMP: Adapting Client/Server Messaging Protocols into Peer-to-Peer Serverless Environments	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Washington,DC,20375**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In this paper we present a generic environment for creating message-oriented server-side proxies to support adaptation from TCP transport-oriented client-server sessions to many- to-many peer-to-peer networking environments more suit- able for deployment in dynamic wireless networks, capable of multicast forwarding. At its input, GUMP provides an in- terface for exposing network server implementations in order to allow existing GUI applications to connect to GUMP. At the back-end, GUMP's generic service discovery and mul- ticast interfaces allow access to multiple implementations enabling the discovery of necessary services on the network maintenance of the network state, and transport of messages amongst peers, for tuning to a speci c network environment. At the heart of GUMP, there is a mechanism for selecting a server-side proxy implementation for a given messaging protocol, allowing multiple proxies to co-exist and run time adaption of the system. As a primary example and use case we show how GUMP has been used to implement an XMPP proxy allowing existing o -the-shelf XMPP client software to dynamically create and operate multi-user chat sessions in a serverless network environment. This resulting proxy integration demonstrates the power of GUMP in its abil- ity to adapt between di erent methods of input using either HTTP or TCP oriented server systems, the use of its di er- ent discovery subsystem bindings (SLPv2 and JmDNS), and its support for multicast architectures. GUMP therefore al- lows a single messaging protocol server-side implementation to be dynamically adapted to suit a particular distributed wireless deployment environment at run time.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**8**	

ple in particular because it makes an explicit assumption on TCP in its protocol design. Whilst XMPP is a very successful open technology, this has often restricted its deployment to conventional network environments that lend well to the TCP transport and server-oriented deployments. Further, other XML-based technologies, such as the WS-* suite from W3C and Oasis, also are typically deployed on stacks built on HTTP, which also has a TCP dependency and a client/server orientation. The translation of such protocols into a more group-oriented or serverless network deployment is non trivial because often the protocols leverage the features of the underlying network stack and/or the availability of fixed network servers in their design, and therefore do not translate well without significant modification.

In this paper, we present a toolkit that supports such network environment adaptivity mechanisms, called the Generic Unicast to Multicast Proxy (GUMP). GUMP exposes network adapters to facilitate the ability to translate between TCP or HTTP connections to corresponding XML-based standardized protocol server proxies, and at the same time allows such implementations to leverage multiple underlying multicast technologies and service discovery subsystems. GUMP allows messages to be transferred between TCP sessions and messaging server proxies to intelligently negotiate protocol-specific adapting issues and maintain the underlying network state within a serverless environment. In this paper, we discuss the motivation, architecture and implementation of GUMP and illustrate its capabilities through the use of a proof-of-concept server proxy implementation of an XMPP adapter to manage XMPP peers on a server-less network environment to enable multiple user chat sessions.

The next section provides a context for GUMP by describing some motivational messaging protocols that we wish to adapt into a serverless environment. Section 3 provides an architecture for the system and Section 3.1 illustrates the use of the architecture by providing a typical usage scenario. Section 4 gives an overview of GUMP's implementation and provides the context for the underlying technologies. Section 5 describes the steps necessary in order to create an XMPP server-side proxy using GUMP and Section 6 discusses the advantages of the approach. In Section 7, we discuss other related approaches around this specific problem domain. In Section 8 we discuss our future directions for GUMP and then in Sections 9 and 10 we conclude the paper and acknowledge our collaborators.

2. GUMP MOTIVATION AND RELEVANCE

This work is motivated by distributed messaging standards, such as XMPP, WS-Notification [4] and WS-Eventing [5], which focus on providing functionality broadly based around chat, presence or asynchronous messaging. These XML-based messaging protocols are based around a client-server concept using management servers for maintaining the state of the network (e.g. for setting up sessions, keep track of which users are logged on, maintaining multi-user chat (MUC) rooms etc). Such a model does not translate well to more transient network deployments as a single point of contact for initiating or managing sessions generally cannot be relied upon. It is therefore advantageous to consider the use of many-to-many transport protocols, such as multicast, in such deployment environments. There are a number of applications that can make use of such a system, not only in our focus area for military applications at the tactical edge, but also in related areas, such as medical emergency scenarios and in disaster response.

However, currently it is quite difficult to adapt applications for use in more mobile, distributed environments because the TCP and client/server dependencies are generally inherent. For example, XMPP can clearly be very useful as a flexible messaging protocol in a mobile application setting to enhance chat functionality, pub/sub capabilities, XML streaming, and presence features. However, at present, there is not a convenient method for mediating this technology into a more distributed wireless environment. Either a new standard would need to be developed or a mapping layer would be needed in oder to adapt the protocol for deployment to the underlying transport stacks and discovery methodologies. Furthermore, WS-Eventing and WS-Notification have similar issues because of their deployment dependency to HTTP and fixed network addresses. The ability to transcend network infrastructure and to connect Grid and mobile hybrid applications transparently in order to run such applications is a goal that spans a number of applications in healthcare, the military, emergency response to name a few, where data is either generated, collected, or exchanged between static nodes and mobile entities.

To realise this potential, we need a capability to support transparent and dynamic access to "serverless" peer-to-peer environments for communication with the mobile domain. This would typically require adapting from a one-to-one connection (TCP) to one-to-many (and many-to-many) relationships, using protocols such as multicast. Such a self-adaptive mechanism is rather complex, involving more than a *simple* transport mapping, because it involves the distributed management of session information across all the peers in the network. Thus, an intelligent self-adaptive mechanism coupled with a supporting underlying adaptive service discovery infrastructure is needed in order to support the discovery of the current state of the network allowing a server proxy to intelligently respond to clients operating within a more connectionless network domain. Such an infrastructure would then enable a XML-based protocol server proxy to answer questions such as "who is around?", "what XMPP MUC groups are available to join?" and "what WS-Notification topics are available?", using standardized discovery protocols, such as the multicast Domain Name Service with Service Discovery extensions (mDNS-SD or Apple bonjour)[6, 7], the Service Location Protocol version 2 (SLPv2) [8], the Simple Service Discovery Protocol (SSDP) in Microsoft's Universal Plug n' Play (UPnP) standard [9], and Bluetooth's Service Discovery Protocol (SDP) [10].

3. GUMP ARCHITECTURE

GUMP provides a convenient framework for allowing developers to create network agnostic server-side proxies for messaging protocols. It provides an environment that allows the developer to focus on the development and detail of the proxy without being tied down to the specifics of how its corresponding entities are to be discovered nor how messages are to be sent between those entities. GUMP also, provides multiple entry points into the system through multiple input bindings of the system, such as TCP and HTTP. Figure 1 shows an overview of the GUMP architecture. At a high conceptual level, GUMP is divided into four main areas:

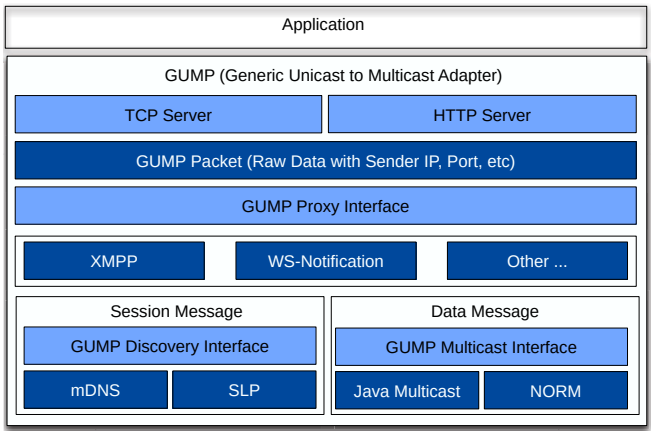

Figure 1: GUMP Architecture

1. **Application connectivity** addresses how applications connect to GUMP through GUMP's *Input Binding*. There are a number of ways of achieving this. Here, we focus on adapting to GUMP at the network level, by exposing GUMP as a network server that accepts arbitrary connections from applications using HTTP or TCP.

2. **Server Proxy** provides a pluggable interface to multiple messaging protocol server proxies, each addressing a specific protocol e.g. XMPP, WS-Notification and so forth. A protocol server proxy translates from a client request into a multicast backbone through the use of the underlying discovery subsystem and GUMP's interface to multicast.

3. **Session Messaging** provides an interface to various discovery subsystems e.g. JmDNS or SLP for use by the proxy to be able to advertise and subsequently discover entities on the network.

4. **Data Messaging** provides an interface to various underlying multicast implementations. Primarily we are focusing on investigating two implementations here: the default Java multicast implementation and NORM (NACK-Oriented Reliable Multicast) [11] transport for reliable multicast delivery.

Together these interfaces provide a strong development environment for creating server proxies for messaging protocols. They allow the adaption of the deployment of a messaging server proxy by facilitating the switching of application connectivity, the ability to employ the use of different discovery algorithms and multiple multicast implementations. This provides the flexibility to be able to experiment with different run-time deployment conditions when applied to different wireless networks. For example, for a non-mobile, relatively benign wireless network, group messaging applications based upon server/client TCP message transport and JmDNS discovery might form an effective solution in terms of overhead and success rates, whereas in MANET environments, more suitable discovery mechanisms (e.g. SLP) and improved multi-hop multicast forwarding with reliable UDP based transport, such as NORM, would result in a more robust dynamic, distributed deployment. The ability to experiment with different configurations is a necessity when dealing with distributed, wireless networks because often the rate of disruption and mobility determines which algorithms are more applicable and one cannot simply apply a one-size-fits-all mentality as one might in more static networks.

The protocol server proxy's role is to translate protocol specific messages from a client, e.g. connect, user arrived, user left, create MUC group, etc into a serverless environment, and subsequently provide an appropriate response back to the client upon completion of its request. The client therefore thinks it is talking to a server using a client/server relationship but in fact for example, it could be talking to a TCP GUMP server that is delegating the requests to a GUMP XMPP protocol server proxy, which is in fact using SLP to discover entities on the network and NORM to pass messages reliably between them, respectively. An example of how a proxy operates in such an environment is provided in the next section.

3.1 GUMP Operation

Focusing at the network level and as shown in Figure 1, GUMP can adapt applications at their transport level either through the use of a locally deployed HTTP or TCP server. For example, rather than an application connecting to an XMPP server, such as OpenFire [12], the user would type in a local address (127.0.0.1:5222), which would re-route the message via this local network GUMP proxy server. The incoming connection requests and messages are then sent directly to the proxy chosen for this instance of GUMP e.g. an XMPP proxy.

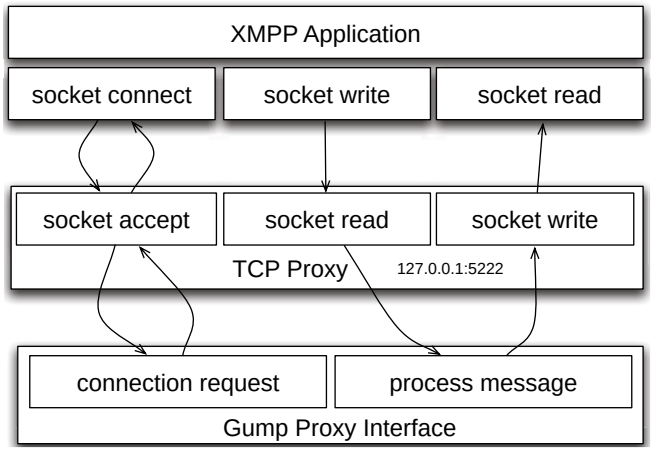

Figure 2: GUMP Input Flow

The semantics of how the connection messages and socket read and write packets are sent and processed by the proxy are provided in Figure 2. Messages sent to the network by the application and received by the GUMP server are wrapped within a GUMP Message and relayed to the GUMP Protocol Proxy. A GUMP message contains a byte array for the data packet, along with information about the sender (host, port) and other related metadata, such as message receiver ID (for supporting multiple connections) and so forth. The Proxy consumes the message and uses the underlying discovery subsystem and multicast to process the message and translate the request into behaviour suitable for the underlying serverless peer-to-peer network.

For example, the request could be an XMPP "create MUC group". The proxy would first search to see if such a group

existed (by using the discovery subsystem for example to query for "Muc Groups"). If a group does not exist, then it would create a MUC group and allocate an underlying multicast group to deal with this traffic. It would then advertise this group to the network, again using the discovery subsystem. Thereafter, if the proxy received a message to be sent to this newly created MUC group, it would look up the multicast address assigned to this group or topic for the various subscribers, and then it would send the message to this group using the GUMP multicast interface, which in turn, would use the underlying multicast transport to send the data onto the network.

4. BACKGROUND AND INTEGRATION OF UNDERLYING TECHNOLOGIES

In this section, we provide a background to the various technologies GUMP integrates and describe their relevance to the deployment within a mobile ad-hoc wireless network (MANET). The system performance challenges of a MANET require far more fault-tolerant distributed service discovery protocol and transport designs than one would expect within more stable network infrastructures. In the following sections, we outline the underlying technologies that GUMP integrates and briefly discuss why such protocols are chosen for deployment for this environment.

GUMP is written in Java and provides a collection of interfaces and factory-based implementations for the various adaptors for GUMP's input, server proxy, discovery and multicast subsystems. The key integration aspects for a MANET's perspective are the discovery and multicast subsystems, which form the basis for locating and communicating with services on the network. In this section, we first provide a brief overview of how applications connect to GUMP server proxies and discuss XMPP to give context for the use-case implementation described in this paper. We then focus in more detail on the discovery and multicast protocols and discuss why these solutions might be appropriate for this type of environment.

4.1 Application Input and Server Proxies

Currently, we have implemented a TCP binding for the GUMP input adapter, which addresses the requirement for the current XMPP server proxy integration, described in Section 5 of this paper. At its core, the Extensible Messaging and Presence Protocol (XMPP) defines a protocol streaming of XML messages. It supports near-real-time messaging, presence, and request-response services and is a derivation of the work performed within the Jabber open-source community but later became an IETF standard [2]. XMPP was designed to be a suitable instant messaging (IM) and presence technology and thus defines specific types of messages (stanzas) for binding, exchanging or publishing information. XMPP employs the use of the simple authentication and security layer (SASL) to provide generalized authentication support for connection-oriented protocols. It can also use transport layer security for on-the-wire message security and uses TCP for the underlying transport. XMPP requires that an implementation supports bi-directional TCP sockets so that a server allows the client to share a single connection and allows multiple bindings on the XMPP default port, which is 5222.

We are also in the process of adapting Cardiff's previous WS-Notification server implementation in WSPeer [13] to work in a serverless mode in GUMP. The WS-Notification implementation, being Web Services based, requires a HTTP binding of the GUMP Input interface as illustrated in Figure 1, which has some subtle extensions we will need to make over TCP. This implementation is currently in progress.

Also for input, we have a socket-level Java adapting implementation that is capable of swapping out the Java TCP socket implementations (java.net.Socket) and replacing this with a GUMP Input conforming implementation so that Java applications can directly plug into GUMP without the need of a network TCP or HTTP server. We already have a prototype implementation for this using a GUMP *SocketImpl* implementation and instantiating it through the use of the Java *SocketImplFactory* factory in *java.net*.

4.2 MANETs and Service Discovery

A MANET environment differs from more conventional stable networks because they have dynamic, sometimes rapidly-changing, random, multi-hop topologies, typically composed of relatively bandwidth-constrained wireless links. The dynamic and multi-hop nature of such networks make it particularly difficult to support robust and efficient routing functionality. There has been extensive research in this area (e.g. [14, 15, 16]) from the military defence, academic, standards, and industry sectors. Although, unicast routing capabilities has been the most investigated, Internet drafts for multicast routing and forwarding have also been suggested [17, 18, 19].

Mobile infrastructure types range from highly autonomous MANET operations to the use of unidirectional satellite links and cellular systems. Discovery mechanism therefore not only have to deal with the core networking issues but also benefit from a variety of operational modalities (unicast, multicast, reactive, i.e. solicited service advert publishing, proactive, i.e. unsolicited service advert publishing) and configuration flexibility (e.g. retransmission timers, data cache settings). In GUMP therefore, we offer flexibility in the choice of discovery implementations, and to this end, we have integrated both JmDNS and SLP for use by the proxies. The implementations are accessed through a two GUMP interfaces: a Java interface called *GUMPServiceRegistration*, which allows services to be registered in a flexible fashion using attributes or a byte array that can contain structured attributes e.g. XML; and a Java abstract class called *GUMPServiceDiscovery*, which allows a client to query and discover a service.

Java Multicast Domain Name Server (JmDNS) [20] is an implementation of multicast DNS in Java. Multicast DNS is a joint effort between the IETF Zero Configuration Networking (zeroconf) and DNS Extensions (dnsext) working groups[1] and provides a way of using familiar DNS programming interfaces, packet formats and operating semantics, in a small network where no conventional DNS server has been installed, through the use of multicast to perform DNS queries. The mDNS protocol, together with DNS Service Discovery, forms the basis for Apple's Bonjour initiative. JmDNS is therefore a well know discovery system and although it might rely rather heavily on the underlying multicast protocol, it forms a good use case for a baseline investigation in a MANET.

[1]Multicast DNS, http://www.multicastdns.org/

The Service Location Protocol (SLP) [21] is an IETF standard for service discovery that allows services to be deployed and discovered with minimal configuration. It was a design predecessor of widely deployed protocols (such as Multicast DNS and DNS Service Discovery) and is one of the several standardized protocols for service discovery. SLP is very interesting to us within the context of MANET for two reasons. First, it employs the combined use of unicast, multicast and distributed directory agent techniques to achieve dynamic deployment and discovery capabilities. Second, it employs the use of a convergence algorithm that provides fault tolerance by repeating service requests according to an exponentially decreasing array of timeouts. Our current customized implementation for SLP also employs the use of a MANET multi-hop forwarding protocol, Simplified Multicast Forwarding (SMF) [19] and dynamic routing algorithms, such as S_MPR [22] and ECDS [23], in its design for more robust MANET routing.

4.3 Multicast Transport

We have implemented a GUMP binding to the default Java multicast and within the context of MANETs, we have also integrated NORM for the reliable delivery of data within collaborating groups. Internet Protocol multicast [24] delivers UDP packets (data) to multiple receivers that have previously joined a multicast group, by efficiently routing and duplicating data at specific routers (chosen algorithmically depending on the particular scheme) that identify more than one receiver downstream in their tree. NORM (an IETF draft [11]) provides fault tolerance to standard multicast by providing end-to-end reliable transport of bulk data objects or streams over generic IP multicast routing and forwarding services. NORM uses a selective, negative acknowledgement (NACK) mechanism for transport reliability and by using limited "a priori" coordination among senders and receivers, it offers additional protocol mechanisms to conduct reliable multicast sessions. NORM incorporates a congestion control algorithm to share available network bandwidth fairly alongside other transport protocols, such as TCP. It is also capable of operating in a unicast mode, which is also of interest to GUMP for allowing the proxies to adapt to other transient connectivity environment that do not support multicast (e.g. for some satellite or cellular connections).

The GUMP binding uses the NORM Java Native interface API and the stream data object model for transmission. Briefly, there are three transport objects (file, data, or stream), which are queued for transmission by NORM senders. The NORM sender controls the transmission rate either manually (fixed transmission rate) or automatically when NORM congestion control operation is enabled. The NORM stream object is more efficient in supporting smaller messages whereas the data object works better for bulkier, more persistent content. Further, in the context of XMPP, a NORM stream can also be *flushed* when a user hits return in their chat application. For Java, the NORM output stream is implemented as a Java java.io.OutputStream and is therefore compatible for use with any of the higher level Java stream classes. NORM input streams, on the other hand, are created dynamically for each NORM sender. Therefore, potentially multiple input streams will be created for each NORM server, which does not map well to the more blocking oriented nature of the core Java stream classes. For the GUMP implementation therefore we created a dispatcher and listener interface for asynchronous notification of events and a *first in first out* blocking queue of *DatagramPacket* objects to queue input messages and identify the sender for each packet. This implementation satisfies the GUMP Multicast socket API and also forms the basis of a NORM Java NIO implementation in the future.

5. DEVELOPING AN XMPP PROXY USING GUMP

As an example of a GUMP plugin for converting a client-server protocol into a peer-to-peer protocol, we present the XMPP Overlay Proxy (XOP). XOP is written to allow group communications using standard XMPP clients with a peer-to-peer protocol on a mobile ad-hoc network (MANET). The clients are "tricked" into thinking that they are communicating with a standard XMPP server that supports Multi-User Chat (MUC) when they connect to the proxy on their local machine. The proxy translates the XMPP messages into XMPP MUC Using Multicast (XMUM), an XMPP-like protocol designed for MANETs, and multicasts them over the network. An XMPP Overlay Gateway (XOG), running on a node which is able to communicate on the MANET and a more traditional network, can transform these XMUM messages back into XMPP MUC messages and relay them to a standard XMPP server somewhere on the traditional network.

The intended use of the XOP is to act as a proxy for an XMPP client located on the same machine. However, it is designed such that it could be used as a proxy for several different clients, so several applications on a node could all use XMPP for communications and share the same XOP instance. Furthermore, in some cases it may make sense for multiple nodes to connect to a remote XOP instance. An example use of this would be cellular phones which connect to an XOP instance located at a mobile base station which is part of a multicast group.

5.1 Connecting the Client to the Proxy

To connect to the proxy, the user configures their client to use *localhost* rather than an XMPP server. The client-server connection usually uses TCP, although the GUMP input binding could be swapped out with a, for example, UDP binding if a UDP XMPP client existed (or was created by embedding GUMP with the client). Once the socket has been created, a client sets up a stream the same as it would with a server. After this, the proxy processes any *presence*, *iq* or *message* messages it receives from the client until the stream is closed.

5.2 Data Messaging

When the proxy receives a *message* or *presence* packet from the client, it passes it to the packet router. The packet router is an application level XMPP stanza/packet router, which routes incoming packets to their representative software endpoints. Examples of endpoints include users locally logged into the XOP instance, a MUC component or other such component, etc. When entities connect to the XOP, the packet router stores routes to them. The packet router determines if the destination is local (a connected entity) or remote. If it is local, it forwards it to the connected entity, otherwise it forwards it to the GUMP multicast interface. Finally, GUMP sends the message out using whatever pro-

tocol (UDP multicast, NORM, etc) it is currently configured to use.

For receiving packets, the XOP creates a GUMP multicast socket to listen for incoming packets. Incoming packets are sent to the packet router, which then forwards them to the appropriate connected entity, if any. The transport mechanism for incoming packets is determined by the GUMP configuration, the XMPP component responsible for listening for packets on the network has no need to know the specific protocol being used.

5.3 Session Messaging

When the proxy receives an *iq* (info/query) packet from the client, it again passes it to the packet router. If it is local, it is forwarded to the connected entity, otherwise it passes it to the IQManager component of the XMPP proxy, which updates a context object if necessary. The context object is intended to help manage the state of the proxy. Keep in mind that with no server, some of the state needs to be managed by the proxy. XMPP context is not used at this point, but could be used in the future, for example to cache request / response pairs. Finally, the IQManager performs the appropriate action (advertise service, reply to the sender with "server information", query for available services, etc) using GUMP's discovery system. GUMP then performs whatever action is needed for the currently configured discovery protocol (SLP, mDNS, etc), and returns an appropriate response to the client if necessary.

The XOP registers a listener with the GUMP discovery interface when it starts up. When a discovery-related event occurs, the XOP formulates an *iq* packet and forwards it to the packet router for delivery to the appropriate entity, or updates the state e.g: adds a MUC room to the available rooms.

5.4 Gatewaying

If the proxy is running on a node with two network interfaces, it can be setup as an XOG. If setup, the XOG can act as a bridge between the peer-to-peer XMUM protocol and the client-to-server XMPP-MUC protocol. The XOG establishes a connection with an XMPP server where the MUC lives using server dialback (see section 8 of [2]), and translates packets between XMUM and XMPP. If a packet arrives on the GUMP discovery interface, the XOG translates it into an *iq* packet and sends it to the receiving server. Likewise, if it receives a presence or message packet from the GUMP multicast interface it forwards it. Vice versa, when an XMPP packet arrives from the receiving server, either the appropriate functions are called in the GUMP discovery interface, or the packets are converted to XMUM and handed to the GUMP multicast interface.

5.5 Theory of Operation

There are several ways the XOP could potentially be used. The first is chat on a MANET, with no external servers. There are only MANET nodes, and they wish to chat as though they were connected to an XMPP MUC server. The XOP provides a way for them to do this with standard XMPP clients.

Another way to use the XOP is for chat on a MANET with a link to a server using the XOG. This allows for a number of MANET nodes to chat with nodes that are not on a MANET, but are traditionally connected to an XMPP MUC chat room. It also allows for groups on entirely different MANETs but with XOGs linking them to the same XMPP server to chat with each other as though they were all in the same MUC room.

As an extension of the previous case, the XOP could also handle cases with a transient connection to the server, providing graceful degradation of capabilities as network capabilities decrease. If the link between the XOG and the server fails, rather than the entire chat infrastructure failing, as would happen with a standard XMPP MUC, the nodes on the MANET will still be able to communicate with each other. The only capability lost is the ability to communicate with the nodes external to the MANET and vice versa.

5.6 Implementation Status

The gatewaying portion of the XOP is still a work in progress. The other features have undergone preliminary testing with both Pidgin[2] and Spark[3]. Both are able to initiate their connections, and send messages as though they were connected to an XMPP server, and the users' experiences should feel no different than when connected to an XMPP server.

6. ADVANTAGES OF GUMP

At its core, GUMP provides a framework for allowing developers to create server-side protocol adapters or proxies for messaging protocols. We specifically target XML-based protocols because that is the main aim of our research but GUMP can be used to integrate other protocols as well. Since such server proxies need to address adapting functions in order to translate from a client/server based connection to a serverless environment, a proxy has to be implemented for each new protocols wishing to be adapted through the use of GUMP. GUMP therefore does not aid in supporting fully automated adaption of any protocol. However, GUMP does provide a convenient environment for focusing on the messaging protocol specific issues without being tied to a fixed architecture or stack of technologies. At the high level and already illustrated through the paper, GUMP provides three key benefits:

1. **Flexible Input Interfaces for Applications:** by defining a generic infrastructure, GUMP allowing multiple input servers or adapters to be plugged in to allow multiple ways of getting data in and out of your proxy implementation. For example, two example implementations shown in figure 1 are to expose a TCP or HTTP server as input, which would allow a broad range of existing applications to connect to the proxy.
2. **Non-Static Discovery Subsystems:** by providing a uniform interface to multiple discovery systems, GUMP allows multiple existing discovery implementations to be plugged in and compared e.g. SLP, Bonjour, JmDNS and so forth.
3. **Choice of Multicast Algorithm:** by providing an interface to multiple multicast algorithm implementations.

These three areas allow any single GUMP server proxy to be deployed in a number of ways using any combination

[2]`http://www.pidgin.im/`

[3]`http://www.igniterealtime.org/projects/spark/index.jsp`

of technologies, simply through configuration. Also, since server proxies are focused at the protocol level, the adaption is not tied to specific GUI applications and can therefore be reused in multiple different ways. Generally (protocol implementation issues aside) only one server-side proxy is needed per protocol (e.g. XMPP, WS-Notification and so forth) and consequently multiple client-side applications can be used without modification with each GUMP server-side proxy. In the context of XMPP, essentially GUMP would allow a developer to create an XMPP serverless server-side implementation, which could effectively replace a server, such as OpenFire, in a wireless network context.

7. RELATED XMPP WORK

XEP-0174 [25] is a Draft Standard of the XMPP Standards Foundation that specifies a link-local messaging protocol defining how XMPP-like communications can be accomplished using zero-configuration networking. This method uses mDNS for service discovery of network entities that support the protocol, including their IP addresses and preferred ports. Any two entities can then negotiate a serverless connection and using XML streams, exchange XMPP messages and IQ stanzas. XEP-0174 is similar to GUMP in that it provides access to underlying discovery protocols but this is achieved through extensions and it is only used to establish one-to-one TCP connections thereafter for the actual communication i.e. a chat.

Jingle (XEP-0166) [26] is a related specification that defines an extension to the XMPP protocol for initiating and managing peer-to-peer media sessions between two XMPP entities. The protocol provides core session management semantics (SIP) to be used for a wide variety of application types (e.g., voice chat, video chat, file transfer) and with a wide variety of transport methods (e.g., TCP, UDP, ICE, application-specific transports). The actual data transfer is delegated to the underlying protocols and therefore message exchange does not form part of the XML XMPP stream.

Neither XEP-0166 or XEP-0174 addresses how one might map from a TCP-oriented connection paradigm to a multicast setting and therefore does not offer serverless group chatting, it rather only offers serverless discovery of the entities and either delegates this to a standard TCP XMPP session (XEP-0174) or to an XMPP out-of-band protocol (XEP-). GUMP is capable of not only initiating an XMPP session, as in XEP-0174, but it can also utilize underlying out-of-band protocols for the communication of actual XMPP stanzas, such as chat messages and so forth, unlike XEP-0166.

XEP-0100 [27] specifies XMPP gateway interaction. The specification defines this to mean gateways that proxy XMPP clients onto non-XMPP servers, such as IRC. This is similar to what is occurring with the XOP, in that it receives XMPP packets and translates them into another protocol. However, there are two main differences. First, the XEP requires a client that implements the XEP registering with a gateway, most likely on an XMPP server such as OpenFire, that has also implemented the XEP. XOP does not require a client or server with these features. Second, the XEP does not support MUC, only one-to-one messaging.

There are a number of middleware solutions for supporting TCP-oriented protocols on dynamic wireless networks, of which Spines [**?**] is one of the more well known examples. It works by creating an overlay routing network and providing a mechanism for hop-by-hop reliability that has been shown to increase reliability and decrease jitter over more traditional overlay routing approaches. This approach differs from GUMP in that it does not modify the protocol itself, it instead makes the existing protocol work better in a dynamic environment.

8. FUTURE WORK

GUMP was designed after a substantial requirements gathering analysis of analysing architectural and deployment considerations of deploying XMPP and WS messaging protocols (WS-Eventing and WS-Notification) onto wireless networks. The resulting framework offers us the ability to not only implement server side proxies for these protocols but it also allows us to compare different multicast algorithms (Java multicast and NORM) and discovery subsystems (SLP and JmDNS) when applied to different mobile environments. The impact of serverless discovery subsystems when applied to frequent mobility of wireless nodes has strong interdependencies with the underlying multicast algorithms they employ. For example, in previous preliminary studies we have seen that S_MPR [22] and ECDS [23] outperform classical multicast flooding approaches for highly mobile scenarios. We would like to investigate the impact of such underlying algorithms when used with high level message protocols, such as WS-Notification and XMPP. Further, we would like to investigate how much protocols such as NORM impact the reliability of the message delivery compared to the standard multicast approach.

We are also investigating the possibility of integrating different unicast algorithms for satellite networks connections for more reliable one-to-one connections, such as R-UDP (Reliable UDP), which forms part of a number of Java Internet applications, including Limewire [28]. GUMP can also be used to adapt from a TCP connection to other unicast protocols as well as the multicast mode, being discussed in this paper.

We are also looking into the possibility of making GUMP completely self-adaptive by using heuristic measurements to analyse/predict the mobility of the nodes in the network at that time and to automatically choose the appropriate deployment stack to address the needs of the network at that time. Through a process of experimentation we hope to extract which protocols are suitable for the various modes and levels of mobility, which can lead directly into a self-adapting system for deployment into multiple network environments.

For the XMPP proxy, we also noted that for scenarios where the link between the XOG and the server was anticipated to phase in and out, some caching feature would be required at the XOG and on the server to store messages meant to go over the link when it is down. This additional persistence of messages is slightly more complicated than the other cases, and from a human-computer interaction perspective it may be useful to provide cues to the user about what the connection status is since there may be a great deal of delay after which a large number of messages would flood the user's screen. This is different from the other intended use cases, which aim to provide an interface for the user that is identical to a standard XMPP MUC experience.

9. CONCLUSIONS

The GUMP framework discussed in this paper is es-

sentially a *means to an end* to enable flexible mappings from TCP and client-server based applications into multiple sender and receiver sessions based on discovery and multicast. GUMP was developed out of a need for being able to transparently translate multiple existing client GUI applications into a wireless network setting. The architecture of GUMP came into being after the realisation that it would be impossible to provide an automatic conversion from TCP to multicast due to protocol-specific behaviour of each protocol. Therefore, rather than attempting to solve a somewhat impractical issue in a generic sense, GUMP takes the approach of providing a framework to make the translation of existing standardised protocols simpler by providing necessary functionality into multiple discovery subsystems, multicast protocols and input adaptors. We have demonstrated in this paper that GUMP has been used to provide a server-side proxy for XMPP, which can adapt between a TCP client connection from an existing XMPP client GUI-based application (Pidgin and Spark have been successfully tested) into a serverless backbone across a wireless network. The power of such a mapping by addressing the mapping at the protocol layer, reduces a potential n-squared application-based approach into a one time protocol mapping, which can support many more client-side protocol compliant applications. The alternative would inevitably lead to the development of further GUIs for specific applications or deployments, which would not only lead to duplication of effort but would result in a far longer development time in the long term.

10. ACKNOWLEDGMENTS

GUMP has been development by the Networks and Communication Systems Branch of the IT Division at NRL. Ongoing modifications to the core system is being funded by the Sonoma project. We thank Andrew Harrison at Cardiff for input in order to support the WS-Eventing/Notification aspect of this work, which is currently under development. We also thank Brian Adamson for his guidance and input into the NORM protocol and its integration.

11. REFERENCES

[1] "Mobile Ad-hoc Networks (MANET)." [Online]. Available: http://www.ietf.org/html.charters/\manet-charter.html

[2] "Extensible Messaging and Presence Protocol (XMPP): Core." [Online]. Available: http://tools.ietf.org/html/rfc3920

[3] "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence." [Online]. Available: http://tools.ietf.org/html/rfc3921

[4] OASIS, "WS-BaseNotification," June 2004, http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf.

[5] D. Box, L. F. Cabrera, C. Critchley, F. Curbera, D. Ferguson, A. Geller, S. Graham, D. Hull, G. Kakivaya, A. Lewis, B. Lovering, M. Mihic, P. Niblett, D. Orchard, J. Saiyed, S. Samdarshi, J. Schlimmer, I. Sedukhin, J. Shewchuk, B. Smith, S. Weerawarana, and D. Wortendyke, "Web Services Eventing (WS-Eventing)," W3C, Tech. Rep., August 2004. [Online]. Available: http://www.w3.org/Submission/WS-Eventing/

[6] M. Giordano, "DNS-Based discovery system in service oriented programming," *Lecture notes in computer science*, vol. 3470, p. 840, 2005.

[7] D. Steinberg and S. Cheshire, *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, Inc., 2005. [Online]. Available: http://portal.acm.org/ft_gateway.cfm?id=1201080&type=safari&coll=GUIDE&dl=GUIDE&CFID=67573037&CFTOKEN=43706727

[8] E. Guttman, "Service location protocol: Automatic discovery of IP network services," *IEEE Internet Computing*, 1999.

[9] S. Helal, "Standards for service discovery and delivery," *IEEE pervasive computing*, pp. 95–100, 2002.

[10] E. Gryazin, "Service discovery in bluetooth," *Group for Robotics and Virtual Reality. Department of Computer Science. Helsinki University of Technology, Helsinki, Finland. Published at NEC CiteSeer, Scientific Literature Digital Library*, 2006.

[11] "Negative-acknowledgment (NACK)-Oriented Reliable Multicast (NORM) Protocol." [Online]. Available: http://www.ietf.org/rfc/rfc3940.txt

[12] "The Java OpenFire XMPP server." [Online]. Available: http://www.openfire.org

[13] A. Harrison and I. Taylor, "WSPeer – An Interface to Web Service Hosting and Invocation," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 4*. IEEE Computer Society, New York, 2005, p. 175a.

[14] S. Basagni, M. Conti, S. Giordano, and I. Stojmenović, *Mobile Ad Hoc Networking: Edited by Stefano Basagni...[et Al.]*. IEEE, 2004.

[15] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*. New York, NY, USA: ACM, 1998, pp. 85–97.

[16] C. Perkins, *Ad Hoc Networking*. Addison-Wesley Professional, 2000.

[17] S. Lee, M. Gerla, and C. Chiang, "On-demand multicast routing protocol," in *proceedings of IEEE WCNC*, vol. 99. Citeseer, 1999, pp. 1298–1302.

[18] T. Camp, J. Boleng, and V. Davies, "Wireless communications and mobile computing (wcmc): Special issue on mobile ad hoc networking: Research, trends and applications; a survey of mobility models for ad hoc network research," vol. 2, no. 5, pp. 483–502, 2002.

[19] J. Macker, W. Chao, and J. Dean, "Simplified multicast forwarding in mobile ad hoc networks," Naval Research Lab Washington DC Information Technology Div, Tech. Rep., 2004.

[20] Arthur van Hoff, "Java implementation of Multicast DNS." [Online]. Available: http://jmdns.sourceforge.net/

[21] "The Service Location Protocol." [Online]. Available: http://www.ietf.org/rfc/rfc2608.txt

[22] "The nrl olsr routing protocol implementation," http://cs.itd.nrl.navy.mil/work/olsr/index.php.

[23] "Optimized link state routing protocol (olsr)," United States, 2003.

[24] "The Multicast RFC." [Online]. Available: http://www.ietf.org/rfc/rfc1112.txt

[25] "Link-Local Messaging - XEP-0174." [Online]. Available: http://xmpp.org/protocols/linklocal/

[26] "Jingle - XEP-0166." [Online]. Available: http://xmpp.org/extensions/xep-0166.html

[27] "Gateway Interaction - XEP-0100." [Online]. Available: http://xmpp.org/extensions/xep-0100.html

[28] "LimeWire." [Online]. Available: http://www.limewire.com